Getting Started with the Thread Profiler

Disclaimer and Legal Information

Contents

About this Guide

This getting started guide takes you through all the basic steps required to prepare your threaded application for data collection, generate and analyze data using the Thread Profiler. Additional details on Thread Profiler are provided in the product's online help.

Thread Profiler supports applications threaded with OpenMP*, Windows* API, or POSIX* threads (Pthreads*). For complete hardware and software requirements, see the product release notes.

Introduction

The Thread Profiler is a plug-in to the VTune™ Performance Analyzer. Use Thread Profiler to locate bottlenecks that are limiting the parallel performance of your threaded software. The Thread Profiler supports applications threaded with OpenMP*, Windows* API, or POSIX* threads (Pthreads*).

About the Thread Profiler

With the Thread Profiler, you can:

- Determine the best sections of code to optimize for sequential performance and for threaded performance.
- Compare scalability across different numbers of processors.
- Compare the performance impact of using different configuration options when your program is run, such as the thread scheduling method (dynamic or not) or the number(s) of threads used to run your application.
- Determine if load imbalance is hurting parallel efficiency.
- Locate synchronization constructs that impact execution time.
- Identify and compare the performance impact of different synchronization methods or different algorithms.
- Collect performance data on a remote Linux* system for analysis on a Windows* system running the VTune™ Performance Analyzer with Thread Profiler.

For threaded applications using Windows* API or Pthreads*, use Thread Profiler to understand the threading patterns in your multi-threaded software by visualizing thread hierarchies and their interactions, and to identify the following types of performance issues:

- Synchronization delays
- Stalled threads
- Excessive blocking time
- Under or over-utilization of processors

About Activities

The Thread Profiler enables you to create Activities in the VTune™ Performance Environment. Activities are at the core of the data-collection process in the VTune analyzer. Within an Activity, you can specify the types of performance data you wish to collect. For each type of performance data, you need to configure the appropriate data collector and define the application/module profiles. These profiles contain information about the application to execute and the modules to analyze.

Running an Activity is analogous to running an experiment. When you run an Activity, the data collectors collect performance data and save it in the Activity results. The results can be viewed later, without having to rerun the Activity. You can also drag and drop different Activity results into a view to compare data from different experiments.

This guide walks you through the creation of an Activity using the Thread Profiler.

Using Thread Profiler

This section offers a general model for using the Thread Profiler to help you identify and locate bottlenecks that may be limiting the parallel performance of your threaded software.

1. Build and Link your Application

OpenMP* Applications

Before you begin, you need to link and instrument your application with calls to the OpenMP* statistics gathering Runtime Engine. The Runtime Engine's calls are required because they collect performance data and write it to a file.

To prepare your code:

1. Compile your application using the Intel® C++ or Fortran Compiler, v7.0 or higher. See the compiler documentation for details.
2. Link your application to the OpenMP* Runtime Engine using the `-Qopenmp_profile` option.

Windows* API or Pthreads* Applications

To begin using Thread Profiler, you must first build your application to accommodate the binary instrumentation that is required for data collection.

To prepare your code:

1. Use a build that includes symbolic information (`-Zi` or `/Zi`) and link with the `/fixed:no` option so that the resulting executable is relocatable.
2. Build your software with thread-safe run-time libraries by using the `-MD` or `-MT` option. For example, to build test.exe with the Microsoft* Compiler, use the following:
   ```
   cl -MD -Zi test.cpp /link /fixed:no
   ```

With instrumentation, calls to the Thread Profiler library are inserted into your code to record calls to threading and synchronization routines.

Remote Data Collection

To collect data on a remote Linux* system, additional setup and configuration is required. See product release notes, online help, and `tprofileFAQ.htm` installed on the Linux* system for complete details.

2. Use the Thread Profiler Wizard

The **Thread Profiler Wizard** enables you to create a new project and Activity with the Thread Profiler collector that gathers data and analyzes your threaded application.

To launch the Thread Profiler Wizard:

1. Start the Thread Profiler.

2. In the **Easy** Start dialog box or from the main menu, click New Project  to open the **New Project** dialog box.
3. From the **Category** drop-down box, select **Threading Wizards**.
4. Select  **Thread Profiler Wizard**.
5. Click **OK**.
6. Under **Threading Type**, select the appropriate threading type for your application: **Windows* API, Pthreads*** or **OpenMP***.
7. Click the  button to browse and select your instrumented executable. After you specify an executable, the working directory defaults to the directory containing the executable you selected.
8. Supply any command line arguments needed to run your application.
9. For OpenMP* software, specify the **Number of Threads**.
10. Click **Finish** to close the dialog box and create the new project and Activity that uses the executable, directory, and command line options you specified.

The Thread Profiler creates an Activity result containing data about factors that may limit the parallel performance of your threaded application.

NOTES

- *Thread Profiler requires multiple processors (either physical processors or processors with Hyper-Threading Technology) to collect meaningful data.*
- *Collect data using a production-sized test problem to obtain an accurate profile.*
- *To collect data on a remote Linux* system, additional setup and configuration is required. See product release notes, online help, and* `tprofileFAQ.htm` *installed on the Linux* system for complete details.*

3. Investigate and Compare Results

If you have successfully run the Thread Profiler Wizard, you should see Activity results in the Thread Profiler viewer. You are now ready to identify and locate bottlenecks that are limiting the parallel performance of your software.

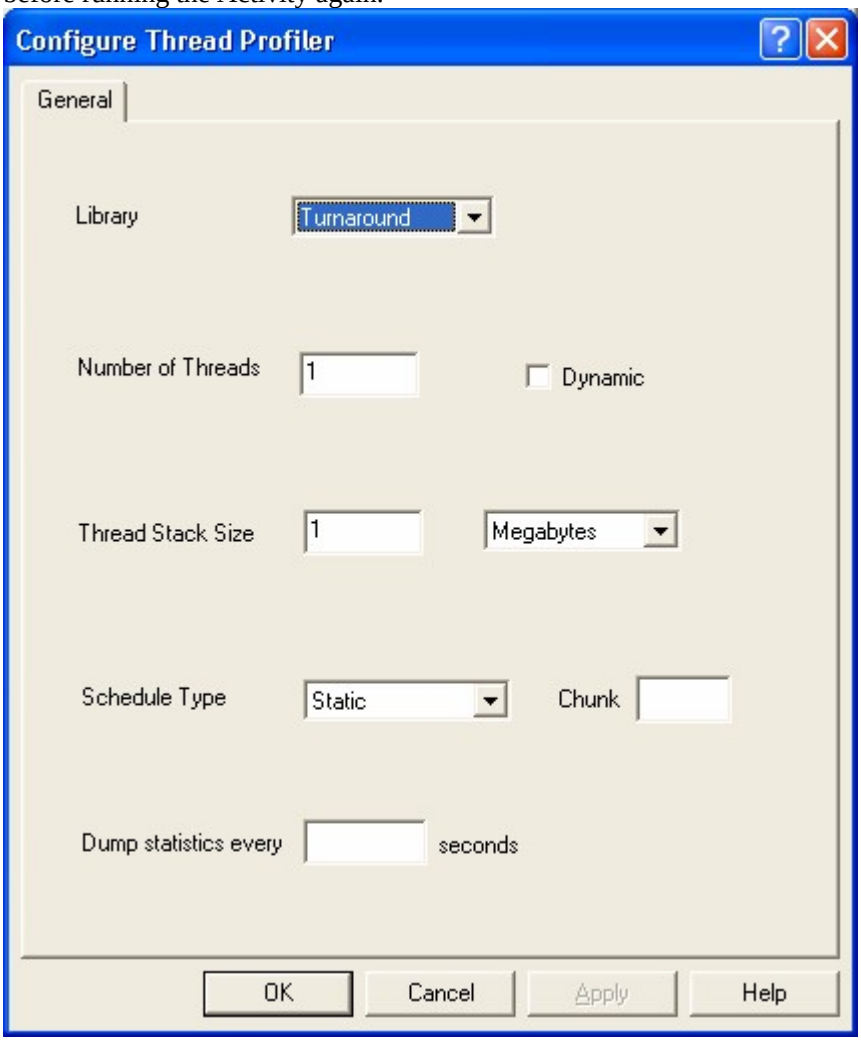
The type of data Thread Profiler provides depend on the type of software you are analyzing. The next sections describe analyzing results for software threaded using one of the following threading methods:

- OpenMP*
- Windows* API or Pthreads*

Analyzing Results for OpenMP*

a) Click the tabs at the bottom of the view to navigate between **Summary**, **Threads**, **Regions**, and **Data** Views.
b) Click the histogram bars to display and study the legend to understand where time was spent by your application.
c) Choose **Configure** > **Modify** > **Modify Selected Activity** to open the **Configure Thread Profiler** dialog box and change various runtime parameters such as runtime scheduling and the number of threads

before running the Activity again.



d) Drag and drop Activity results from the **Tuning Browser** onto the Thread Profiler viewer (bar graph) to compare results. After you drop the Activity result additional bars appear in the graph representing the additional Activity result. Results are shown side-by-side with the bars that were already present from the previous Activity results.

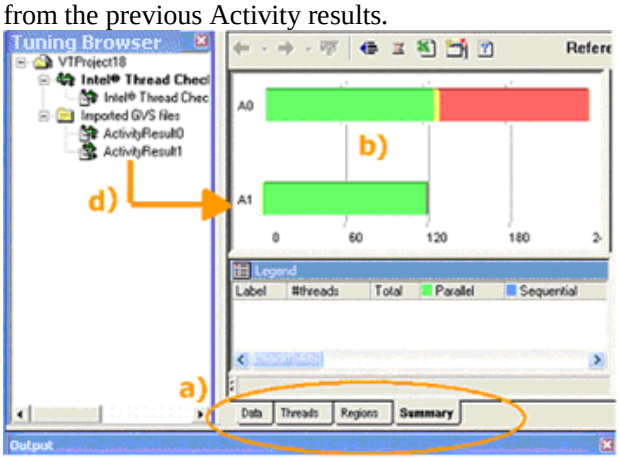


Figure 1: Summary view.

e) Estimate potential speedup by studying the projections in the speedup plot. Applications that are parallelized properly are more likely to scale well (as in Figure 2 than applications that are not well parallelized (as in Figure 3). You

can see in Figure 2 that almost all time is parallel time and as more processors become available the speedup graph projects nearly ideal speedup. In contrast, Figure 3 shows a projected speedup (the red line) of only a bit more than 2x when four processors are used because of lock imbalance.

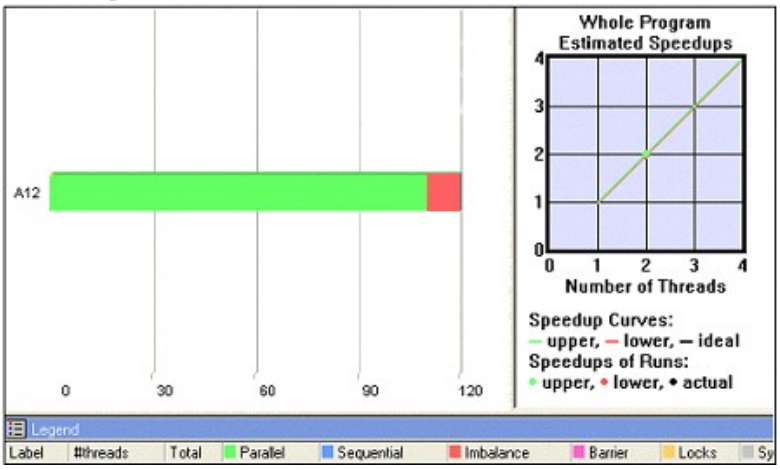


Figure 2: An Activity result for software with good parallelization shows little opportunity for speedup.

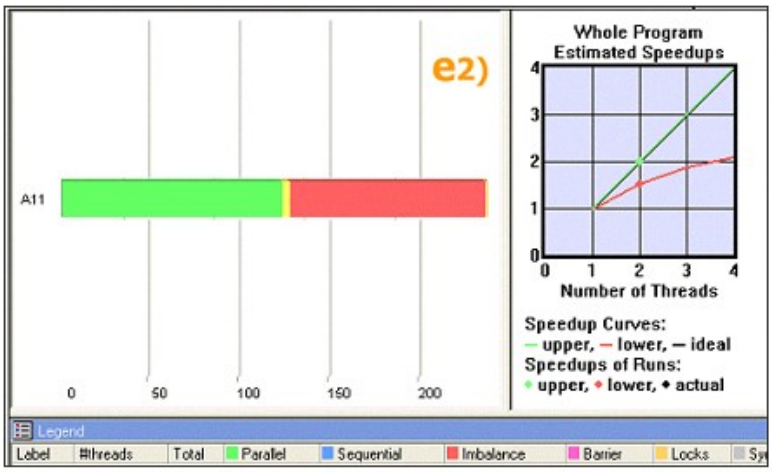


Figure 3: An Activity result for software with poor parallelization shows much opportunity for speedup.

Analyzing Results for Windows* API or Pthreads*

Thread Profiler provides different views for examining and understanding threading behavior of an application. It enables you to analyze threading issues that affect performance and to better understand the nature of the program's threading and synchronization. These issues include synchronization, threading overhead, blocking calls that delay threads, and under- or over-utilizing processors in the system and relate it back to the original application source code. Use Thread Profiler to:

- Group performance data according to critical path, thread, software object, concurrency level, or source location.
- Select one or more bars from a bar chart and filter away unselected bars.
- Sort data according to impact, cruising, overhead, and blocking time. You can also sort by concurrency level (the number of active threads), location in source code, module id, thread id, object id, and object type.

- Identify source locations in your code that cause performance problems.

To navigate between views, click on the corresponding tab.

Thread Profiler Views

The following table summarizes the functions of the Thread Profiler views for analysis of threaded applications:

View	Description
Critical Path	Default view. Displays all critical paths of an execution, decomposed into four major time components: cruise, impact, block, and overhead.
Profile	Displays a high-level summary of the time along at least one selected critical path of an execution.
Timeline	Represents the lifetime of a thread, showing its start time and end time.
Source Views	Show source code locations associated with threading events that impact performance. Creation Source views shows where an event began; Transition Source View shows where an event changed.

4. Optimize Your Code

Code optimization is an iterative process. After analyzing results obtained with Thread Profiler, you modify your code to improve performance. You then analyze your modified code using Thread Profiler, closing in on outstanding performance issues. Repeat the process until you are satisfied that you have eliminated as many performance issues as possible.

Index